New Algorithmic Development on GPU and applications

Viktor K. Decyk and Tajendra V. Singh

UCLA

Abstract

The best algorithms for Particle-in-Cell codes on GPUs partition the spatial domain into tiles with a small number of grid points. On architectures such as the NVIDIA C1060, using one thread per tile avoided data collisions and gave the best performance. Newer NVIDIA Fermi GPUs, however, have cache and fast native atomic updates and using one thread block per tile can give better performance. With this scheme, we have implemented a simple 2D electrostatic skeleton code on the Fermi M2090, using MPI to control multiple GPUs and achieved a performance of 500 ps/particle/time step on 3 GPUs.

2D Electrostatic Skeleton Particle-in-Cell Code on multiple GPUs

Original Implementation on Tesla C1060
New Implementation on Fermi M2090
Implementation on multiple GPUs connected with MPI

GPUs are graphical processing units which consist of:

- 12-30 SIMD multiprocessors, each with small (16-48KB), fast (4 clocks) shared memory
- Each multi-processor contains 8-32 processor cores
- Large (0.5-6.0 GB), slow (400-600 clocks) global shared memory, readable by all units
- No cache on some units
- **Very fast (1 clock) hardware thread switching**

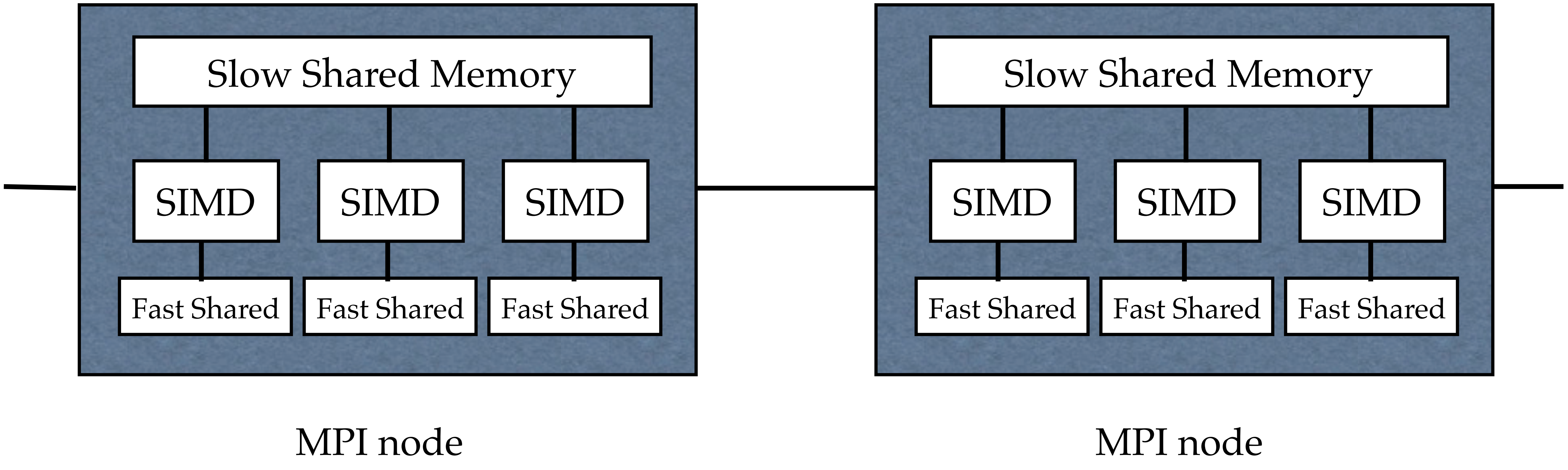
GPU Technology has two special features:

- High bandwidth access to global memory (>100 GBytes/sec)
- Ability to handle thousands of threads simultaneously, greatly reducing memory "stalls"

Challenges:

- High global memory bandwidth is achieved mainly for stride 1 access
  (Stride 1 = adjacent threads read adjacent locations in memory)
- Best to read/write global memory only once

Simple Hardware Abstraction



A distributed memory node consists of

- SIMD (vector) unit works in lockstep with fast shared memory and synchronization
- Multiple SIMD units coupled via “slow” shared memory and synchronization

Distributed Memory nodes coupled via MPI

Memory is slower than computation, and best accessed with stride 1 addressing

- Streaming algorithms (data read only once) are optimal

Particle-in-Cell Codes

Simplest plasma model is electrostatic:

1. Calculate charge density on a mesh from particles:

$$\rho(\boldsymbol{x}) = \sum_i q_i S(\boldsymbol{x} - \boldsymbol{x}_i)$$

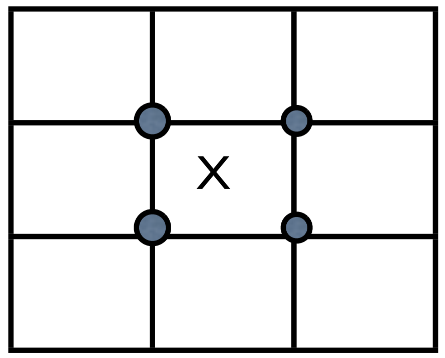
2. Solve Poisson's equation:

$$\nabla \bullet \boldsymbol{E} = 4\pi\rho$$

3. Advance particle's co-ordinates using Newton's Law:

$$m_i \frac{d\boldsymbol{v}_i}{dt} = q_i \int \boldsymbol{E}(\boldsymbol{x}) S(\boldsymbol{x}_i - \boldsymbol{x})\, d\boldsymbol{x} \qquad \frac{d\boldsymbol{x}_i}{dt} = \boldsymbol{v}_i$$

Inverse interpolation (scatter operation) is used in step 1 to distribute a particle's charge onto nearby locations on a grid.



Interpolation (gather operation) is used in step 3 to approximate the electric field from the grid points near a particle's location.

Designing New Particle-in-Cell (PIC) Algorithms

Most important bottleneck is memory access

- PIC codes have low computational intensity (few flops/memory access)
- Memory access is irregular (gather/scatter)

PIC codes can implement a streaming algorithm by keeping particles ordered by small tiles

- Minimizes global memory access since field elements need to be read only once.
- Cache is not needed, gather/scatter can be avoided.
- Deposit and particles update can have optimal stride 1 access.
- Single precision can be used for particles

Designing New Particle-in-Cell (PIC) Algorithms

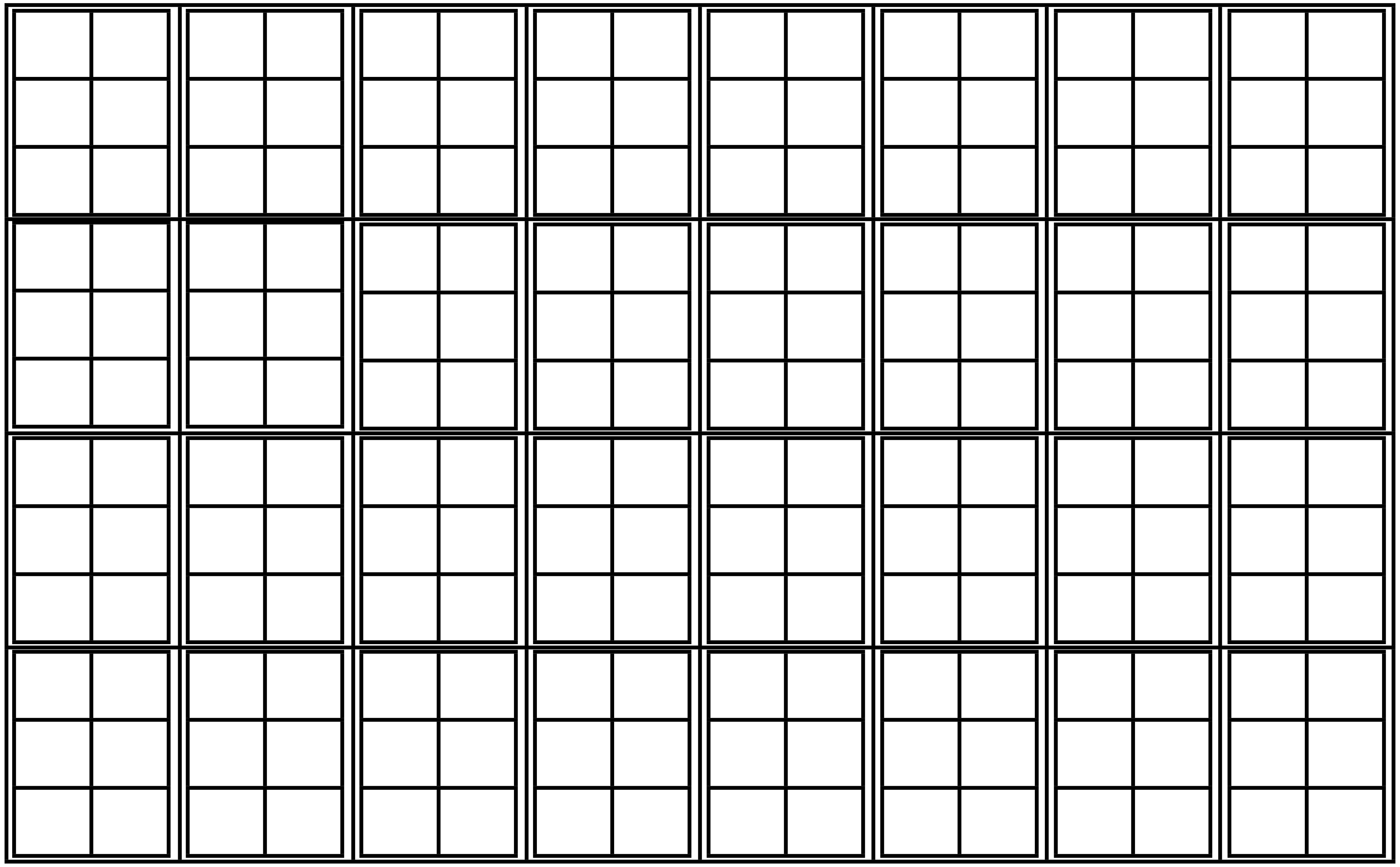
Particles ordered by small tiles, typically 2 x 3 grid points

Original scheme on NVIDIA Tesla C1060 GPU:

- Associate a **thread** with each tile and particles located in that tile
- Tile size is determined by amount of fast memory available

We created a new data structure for particles, partitioned among threads:

```
    dimension part(block_size,idimp,npmax,num_blocks)
```



Designing New Particle-in-Cell (PIC) Algorithms: **Push/Deposit Procedures**:

Within a tile, all particles read or write the same fields.

- Before pushing particles, copy fields to fast memory
- After depositing charge to fast memory, write to global memory

Each thread contains data for its grids in the tile, plus guard cells: an extra grid on the right, and an extra row of grids on the bottom for linear interpolation.

Different threads never write to same memory

Parallelization is easy, each particle is independent of others, no data hazards

- Similar to MPI code, but with tiny partitions

Designing New Particle-in-Cell (PIC) Algorithms: **Maintaining Particle Order**

Three steps:
1. Particle Push creates a list of particles which are leaving a tile
2. Using list, each thread places outgoing particles into a buffer it controls
3. Using lists, each tile copies incoming particles from buffers into particle array

- Less than a full sort, low overhead if already ordered
- Essentially message-passing, except buffer contains multiple destinations

In the end, the particle array belonging to a tile has no gaps
- Particles are moved to any existing holes created by departing particles
- If holes still remain, they are filled with particles from the end of the array

The reordering algorithm does not match the architecture well
- Does not have stride 1 access (poor data coalescing)
- Does not run in lockstep (has warp divergence)

Evaluating New Particle-in-Cell (PIC) Algorithms on GPU: Electrostatic Case

Porting these subroutines to the GPU required:

- Main code written in Fortran90
- GPU code written in Cuda C
- Original Fortran program also ran to validate GPU results

2D ES Benchmark with 256x512 grid, 4,718,592 particles, 36 particles/cell

Original Code ran on 2.66 GHz Intel i7 (Nehalem) Host (Macintosh Pro), using gfortran.

Optimal parameters were blocksize = 32, optimal tile size = 2x3

Evaluating New Particle-in-Cell (PIC) Algorithms on GPU: **2D Electrostatic Case**

```
Hot Plasma results with dt = 0.1
              CPU:Intel i7   GPU:Tesla C1060
Push            18.9 ns.       770 ps.
Deposit          8.7 ns.       260 ps.
Reorder          0.4 ns.       810 ps.
Total Particle  28.0 ns.      1830 ps.

The time reported is per particle/time step.
The total speedup on the Telsa C1060 was 15x.


Cold Plasma (asymptotic) results with vth = 0, dt = 0.025
             CPU:Intel i7   GPU:Tesla C1060
Push            18.6 ns.       560 ps.
Deposit          8.5 ns.       230 ps.
Reorder          0.4 ns.        40 ps.
Total Particle  27.5 ns.       820 ps.

The time reported is per particle/time step.
The total speedup on the Telsa C1060 was 30x.
```

Reference:
V. K. Decyk and T. V. Singh, “Adaptable Particle-in-Cell Algorithms for Graphical Processing Units,” Computer Physics Communications, 182, 641, 2011.

Evaluating New Particle-in-Cell (PIC) Algorithms on GPU: **2-1/2D Electromagnetic Case**

Relativistic Boris mover, deposit both current and charge, reorder twice per time step

Optimal parameters were blocksize = 64, optimal tile size = 1x2

```
Warm Plasma results with c/vth = 10, dt = 0.04
             CPU:Intel i7   GPU:Tesla C1060
Push             81.7 ns.       1.13 ns.
Deposit          40.7 ns.       1.06 ns.
Reorder           0.5 ns.       1.13 ns.
Total Particle 122.9 ns.       3.32 ns.

The time reported is per particle/time step.
The total speedup on the Telsa C1060 was 37x.
```

```
Cold Plasma (asymptotic) results with vth = 0, dt = 0.025
            CPU:Intel i7   GPU:Tesla C1060
Push             78.5 ns.       790 ps.
Deposit          37.3 ns.       820 ns.
Reorder           0.4 ns.       160 ns.
Total Particle 116.2 ns.       770 ns.

The time reported is per particle/time step.
The total speedup on the Telsa C1060 was 66x.
```

Designing New Particle-in-Cell (PIC) Algorithms

NVIDIA Fermi architecture has new hardware.

- Cache now available
- Native floating point atomic addition now supported

This has substantial impact on optimal PIC algorithms

- Multiple threads can safely and quickly update the same memory location
- Naive, simpler algorithms now work much better

A reinvestigation of PIC algorithms on Fermi showed:

- Use of tiles is still optimum
- Assigning a block of threads to each tile rather than one thread improves performance
- Shared memory requirements per thread block is reduced
- Larger tiles reduced cost of reordering

Designing New Particle-in-Cell (PIC) Algorithms

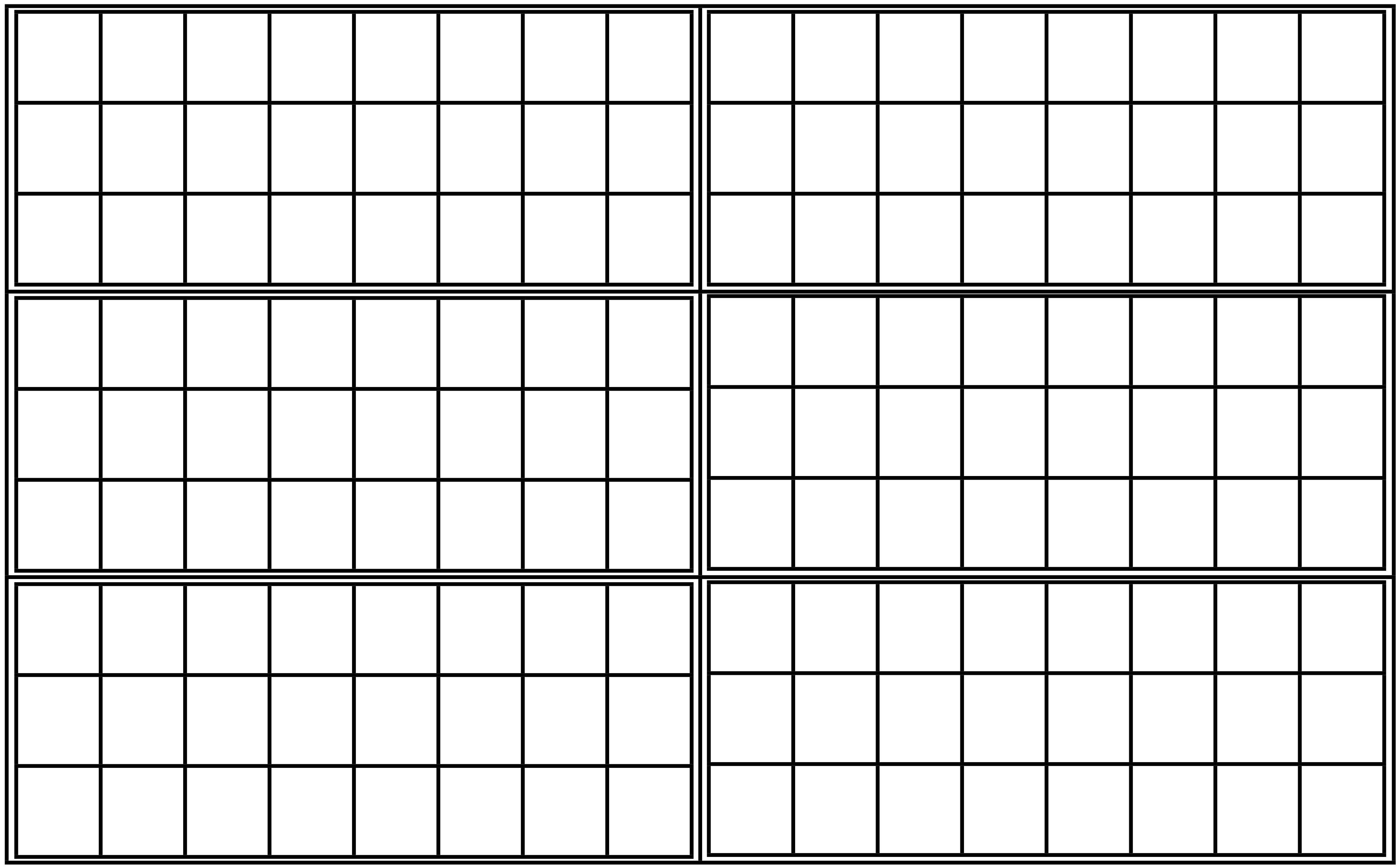
Particles ordered by larger tiles, typically 16 x 16 grid points

New scheme on Fermi M2090:

- Associate a **thread block** with each tile and particles located in that tile

We created a new data structure for particles, partitioned among threads blocks:

```
    dimension part(npmax,idimp,num_blocks)
```



Designing New Particle-in-Cell (PIC) Algorithms:

Deposit now permits different threads to write to the same memory location

- Fast native atomic additions are crucial.

Parallelization is still easy, each particle is independent of others

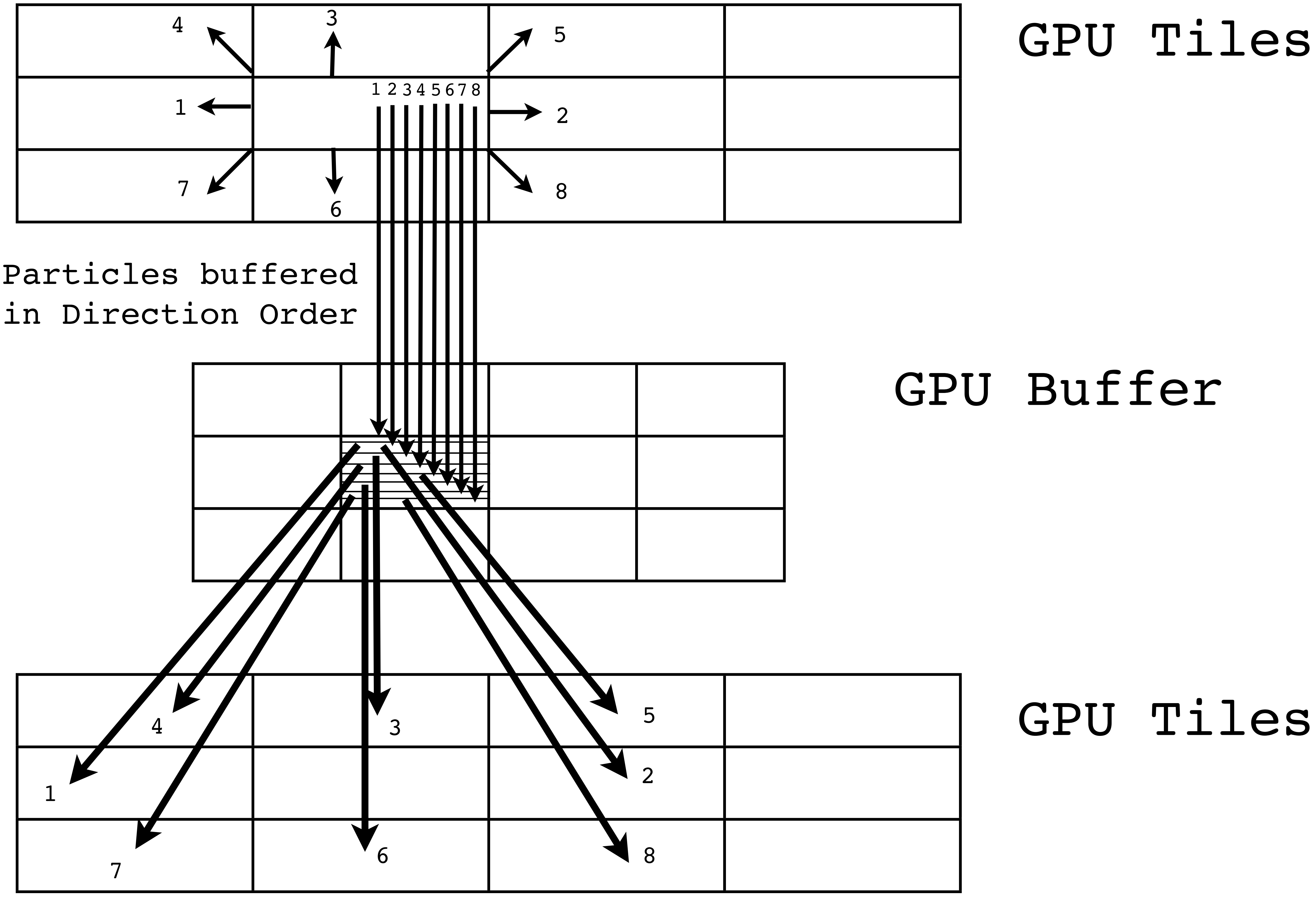
- Deposit now has data hazards, different from MPI code

Maintaining particle order has same three steps:

- But much more difficult to write in parallel
- Atomic operations now needed

Maintaining particle order also improved by creating a table of outgoing/incoming particles

GPU Particle Reordering



Evaluating New Particle-in-Cell (PIC) Algorithms on GPU: **Electrostatic Case**
2D ES Benchmark with 512x512 grid, 9,437,184 particles, 36 particles/cell
(Field Solver not yet implemented with the new scheme)

Hot Plasma results with dt = 0.1

	CPU:Intel i7	GPU:Fermi M2090	GPU:Tesla C1060
Push	18.9 ns.	557 ps.	735 ps.
Deposit	8.7 ns.	254 ps.	232 ps.
Reorder	0.4 ns.	134 ps.	818 ps.
Total Particle	28.0 ns.	944 ps.	1785 ps.

The time reported is per particle/time step.
The total speedup on the Fermi M2090 was 30x,
on the Telsa C1060 was 16x.

Cold Plasma (asymptotic) results with vth = 0, dt = 0.1

	CPU:Intel i7	GPU:Fermi M2090	GPU:Tesla C1060
Push	18.6 ns.	415 ps.	631 ps.
Deposit	8.5 ns.	178 ps.	217 ps.
Reorder	0.4 ns.	18 ps.	23 ps.
Total Particle	27.5 ns.	611 ps.	871 ps.

The time reported is per particle/time step.
The total speedup on the Fermi M2090 was 45x,
on the Telsa C1060 was 32x.

Original scheme performed better on C1060 and with cold plasma on both architectures:
- Repeated atomic updates to the same location was slow

Dawson2 at UCLA: 96 nodes, ranked 384 in top 500, 70 TFlops on Linpack

- Each node has: 12 Intel G7 X5650 CPUs and 3 NVIDIA M2090 GPUs.
- Each GPU has 512 cores: total GPU cores=147,456 cores, total CPU cores=1152

Designing New Particle-in-Cell (PIC) Algorithms: **Multiple GPUs**

Multiple GPUs on one node can be controlled either with OpenMP or MPI

We started with an existing 2D Electrostatic MPI code from UPIC Framework

- Replacing MPI push/deposit with GPU version was no major challenge

With multiple GPUs, we need to integrate two different partitions

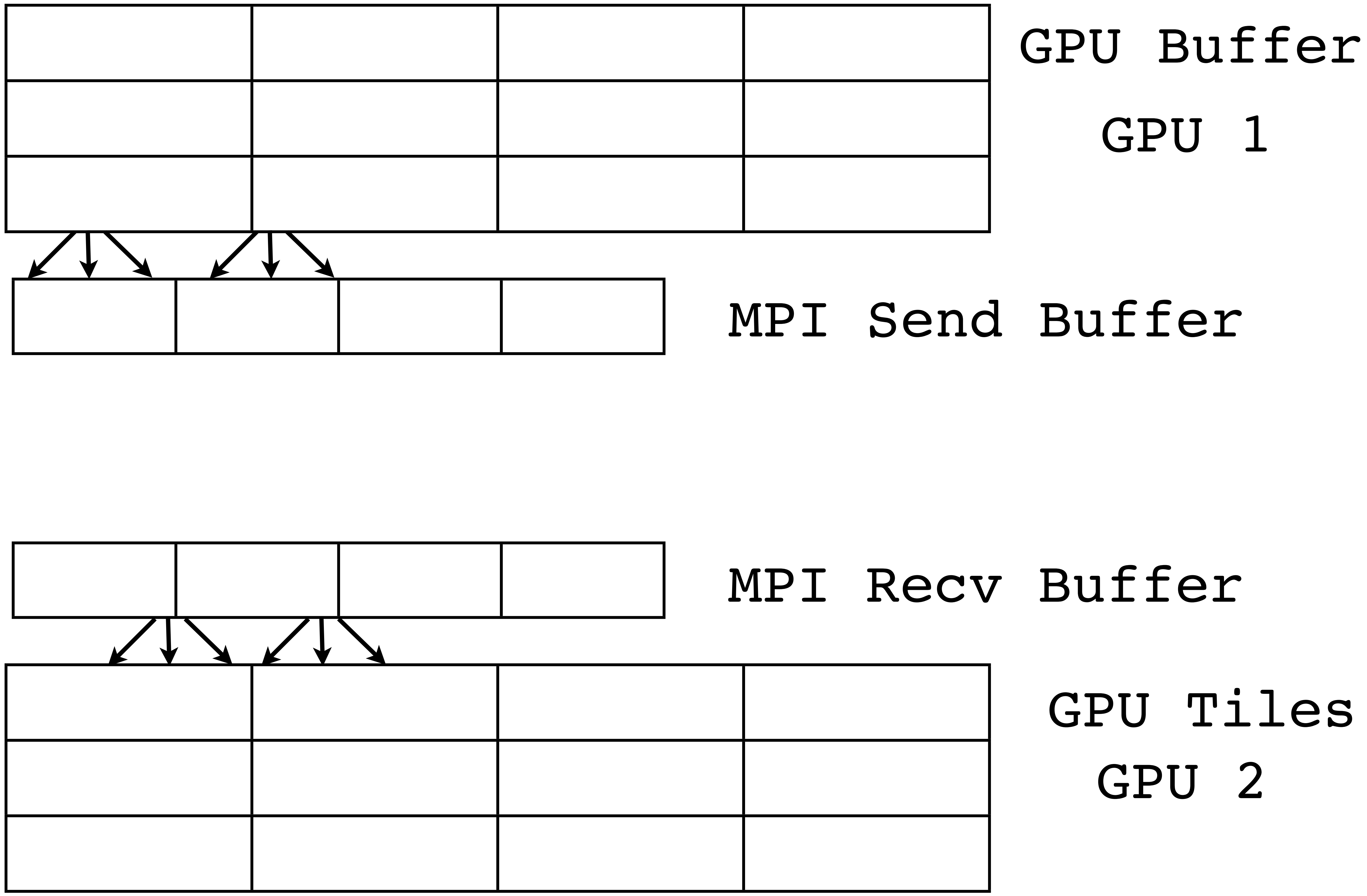
- MPI and GPU each have their own particle managers to maintain particle order

Only the first/last row or column of tiles on GPU interacts neighboring MPI node

- Particles in row/column of tiles collected in MPI send buffer
- Table of outgoing particles are also sent
- Table is used to determine where incoming particles must be placed

Field solver still performed on host

GPU-MPI Particle Reordering



Evaluating New Particle-in-Cell (PIC) Algorithms on GPU: Electrostatic Case
2D ES Benchmark with 512x512 grid, 9,437,184 particles, 36 particles/cell
(Field Solver not yet implemented on GPUs)

```
Hot Plasma results with dt = 0.1
CPU:Intel i7    1 core         12 cores
Push            20.30 ns.       1.80 ns.
Deposit          8.34 ns.       0.75 ns.
Reorder          0.34 ns.       0.04 ns.
MPI Move         0.01 ns.       0.04 ns.
Total Particle  28.94 ns.       2.64 ns.

GPU:Fermi M2090  1 GPU             3 GPUs
Push             345 ps.           135 ps.
Deposit          266 ps.            97 ps.
Reorder          478 ps.           187 ps.
MPI Move          36 ps.            88 ps.
Total Particle  1125 ps.           506 ps.


The time reported is per particle/time step.
The total speedup on the 3 Fermi M2090s compared to 12 cores was 5.2x,
Speedup on 3 M2090s compared to 1 M2090 was 2.2x
```

Conclusions

PIC Algorithms are largely a hybrid combination of previous techniques
- Vector techniques from Cray
- Blocking techniques from cache-based architectures
- Message-passing techniques from distributed memory architectures

Scheme should be portable to other architectures with similar hardware abstractions

2D Electrostatic code is a very simple code, with low computational intensity
- only 55 Flops per particle update

PIC codes with higher computational intensity should do better
- EM codes, 3D codes, higher order interpolations

Fermi M2090 is about 2x faster than the C1060
Expect 2-1/2D EM code to run about 1.5 ns/particle/time step

Further information available at:
http://www.idre.ucla.edu/hpc/research/